

Neuronal and Evolutionary Computation

(MIA-MEISISI) Practical Exercise

A2 Supervised Learning

**Students Name: MD ARIF ULLAH BHUIYAN & JULIO CÉSAR SIGUEÑAS
PACHECO**

Github Link: <https://github.com/arifbhuiyan82/Classification-with-SVM-BP-and-MLR>

Objective: Perform data classification with the following algorithms:

Support Vector Machines (SVM), using free software

Back-Propagation (BP), using free software

Multiple Linear Regression (MLR), using free software

Support Vector Machines (SVM): Support Vector Machines (SVM) are a type of supervised machine learning algorithm mainly used for classification tasks, but can also handle regression. They work by finding a hyperplane that best separates different classes of data points. SVMs are particularly effective in high dimensional spaces and can handle both linear and non-linear data, thanks to the use of kernel functions. They are known for being memory efficient and versatile, but require careful tuning of parameters and are less effective when the number of features significantly exceeds the number of samples. SVMs are widely used in applications like image and text classification.

Support Vector Machines (SVM), using free software: For implementing Support Vector Machines (SVM), we are going to use a free software called Weka (Waikato Environment for Knowledge Analysis). Weka is a popular suite of machine learning software written in Java, developed at the University of Waikato in New Zealand. It is a comprehensive toolkit for performing various tasks in data mining and machine learning.

First of all, we install the package called LibSVM (Figure 1). LibSVM is a separate and widely-used library for Support Vector Machines (SVMs) that provides an efficient and flexible platform for SVM classification and regression.

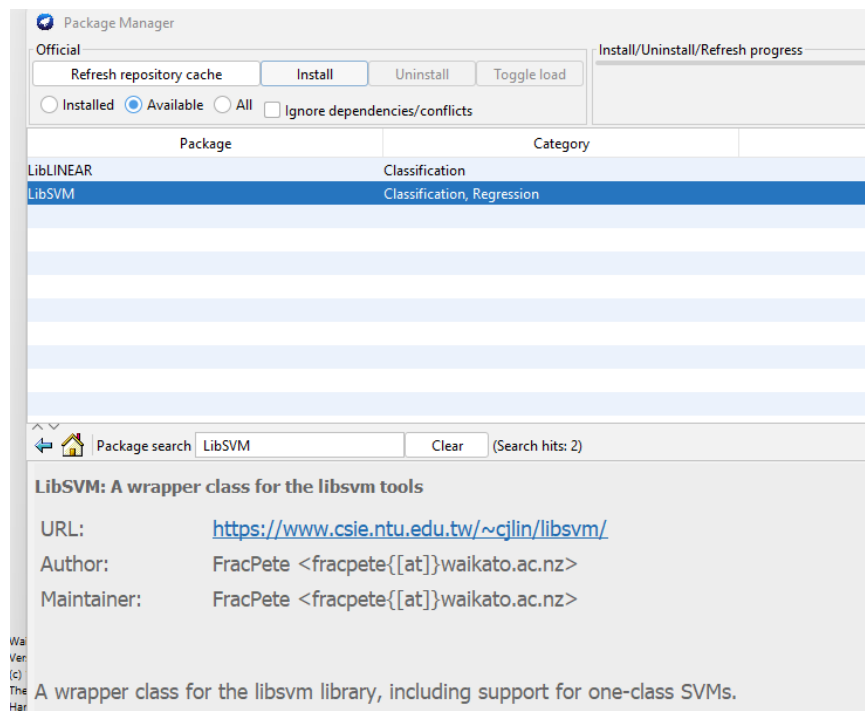


Figure 1: Install LibSVM package in Weka

Since this package does not support numerical values as class labels, it is necessary to prepare a CSV file containing the data, replacing the numbers in the final column with 'A' and 'B' in place of 0 and 1, respectively (as shown in Figure 2).

A2-ring-separable.txt - Notepad

FileEditFormatViewHelp

-0.137094	0.899654	0
0.542574	-0.492435	1
-0.658204	0.605110	0
-0.869820	0.519287	0
0.678610	0.346579	1
0.186743	-0.385230	1
0.254768	-0.651996	1
-0.366600	-0.865114	0
0.134406	-0.343083	1
-0.014929	0.150961	0
0.325971	-0.992276	0
0.963842	0.093940	0
-0.906254	-0.158387	0
-0.493524	-0.547124	1
-0.106125	-0.806331	0
0.220624	-0.883067	0
0.598772	-0.249684	1

Paste

Clipboard

Font

Alignment

A1

X✓fx

Input1

	A	B	C	D	E	F	G
1	Input1	Input2	Output				
2	-0.13709	0.899654	A				
3	0.542574	-0.49244	B				
4	-0.6582	0.60511	A				
5	-0.86982	0.519287	A				
6	0.67861	0.346579	B				
7	0.186743	-0.38523	B				
8	0.254768	-0.652	B				
9	-0.3666	-0.86511	A				
10	0.134406	-0.34308	B				
11	-0.01493	0.150961	A				

Figure 2: Replacing the numbers with 'A' and 'B' in place of 0 and 1

Next, we upload the "A2-ring-separable-letters.csv" file into the Weka Explorer (as shown in Figure 3)

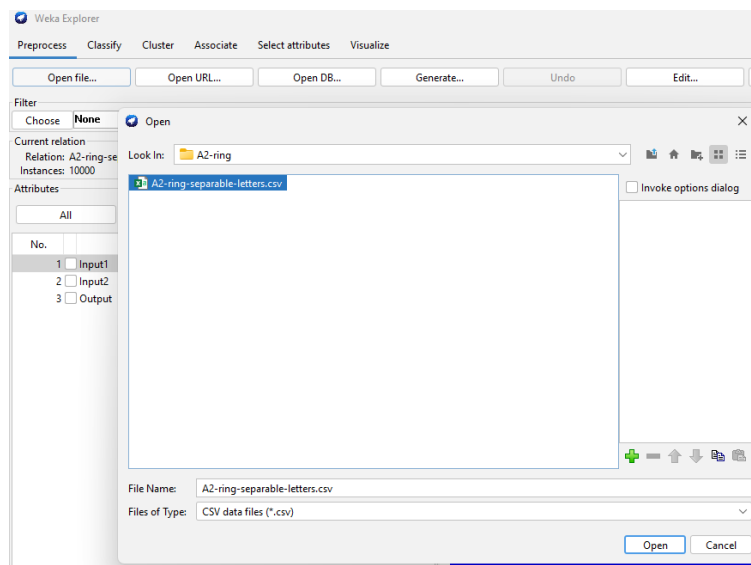


Figure 3: Uploading the "A2-ring-separable-letters.csv" file

And then navigate to the "Classify" tab. Here, we proceed with the learning process using only the training set. But here we chose LibSVM as a classifier. (as shown in Figure 4)

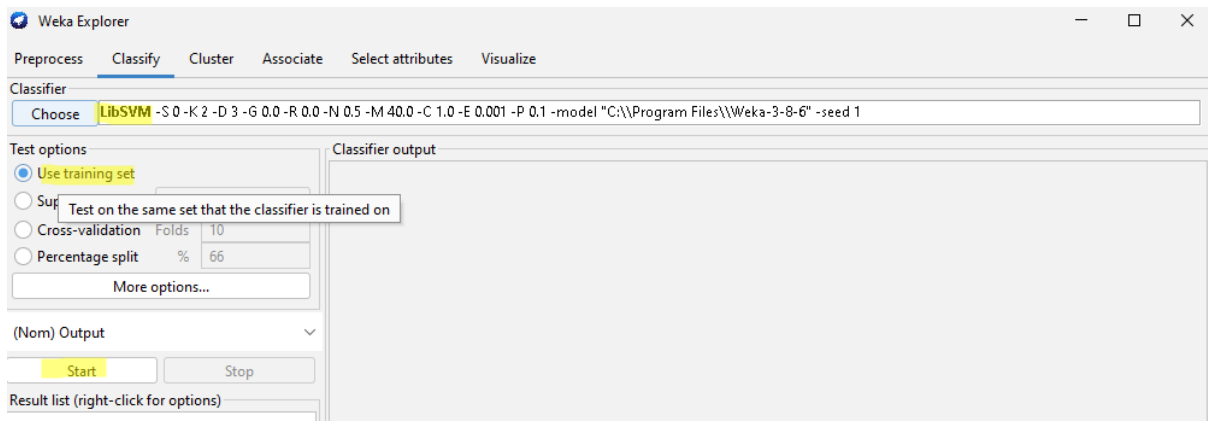


Figure 3: SVM only using training set (separable).

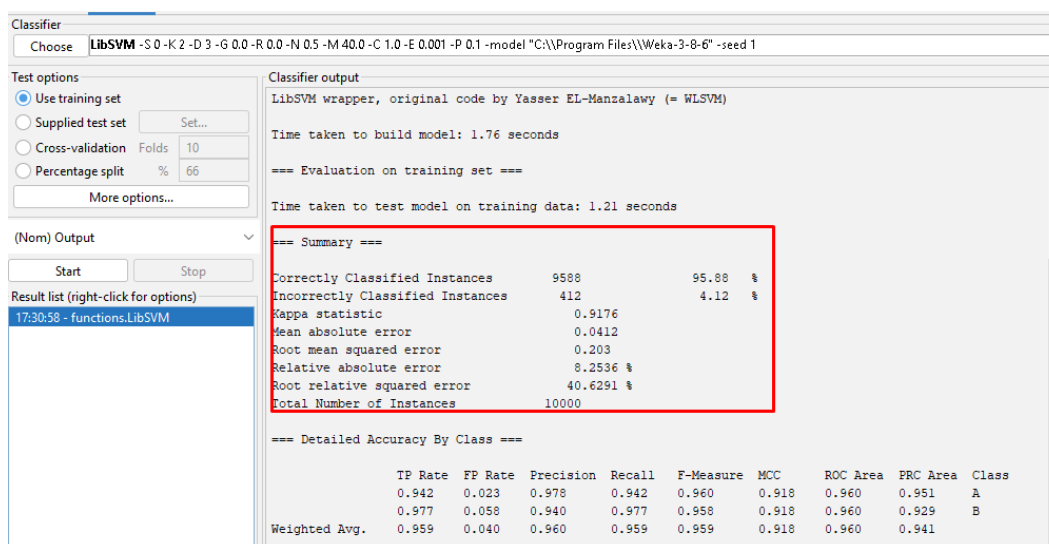


Figure 4: Summary of SVM only using training set

Now we upload "A2-ring-test-letters.csv" as a Supplied test set and begin the training.

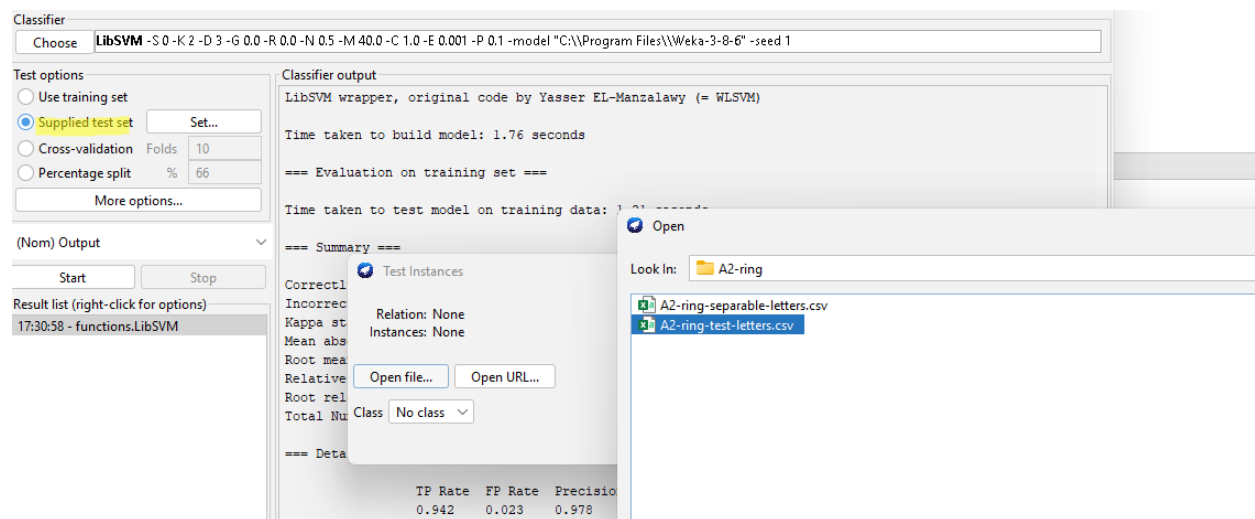


Figure 5: SVM adding test set (separable)

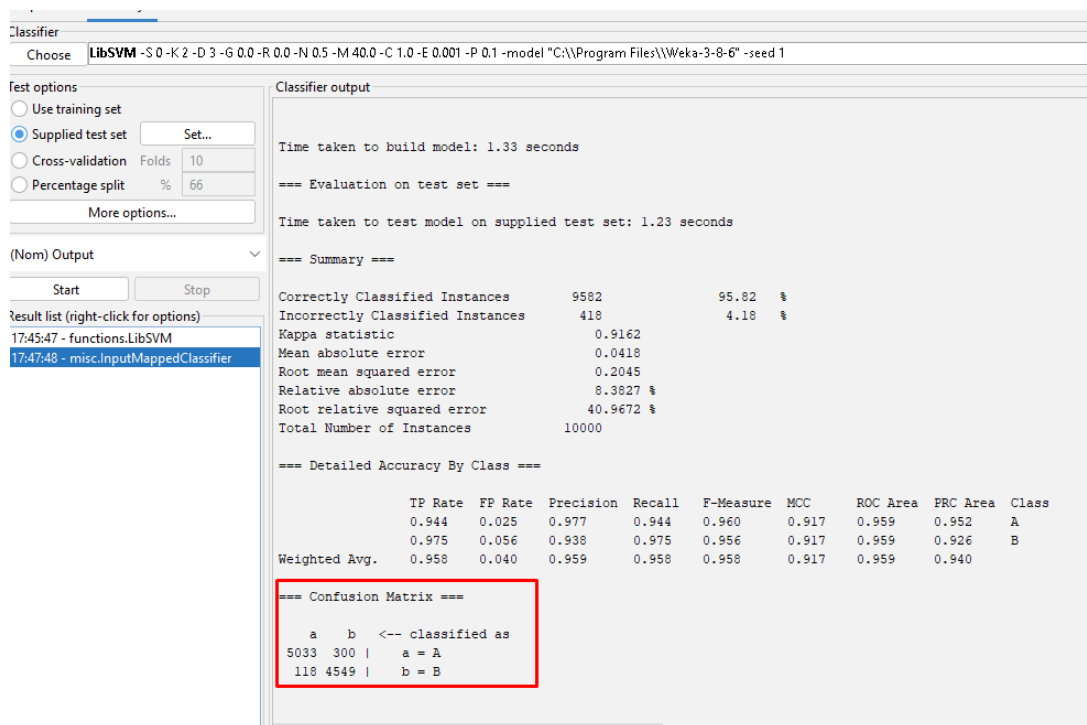


Figure 6: SVM train result using test set (separable)

Having the confusion matrix in Weka we can easily compute the classification error according to the formula where-

$n_{01} = 300$ # False positives

$n_{10} = 118$ # False negatives

$n_{00} = 5033$ # True negatives

$n_{11} = 4549$ # True positives

Classification error formula:

$$E = 100 * (n_{01} + n_{10}) / (n_{00} + n_{01} + n_{10} + n_{11})$$

$$E = 100 (300 + 118 / 5033 + 300 + 118 + 4549)$$

$$E = 100 (418 / 10000)$$

$$E = 100 * 0.0418 = \mathbf{4.18\%}$$

In Weka, detailed information about classification errors can be accessed by navigating to More Options, selecting Output Predictions, and then choosing the CSV format. This process yields an output of 10,000 actual values alongside the predictions made by the software.

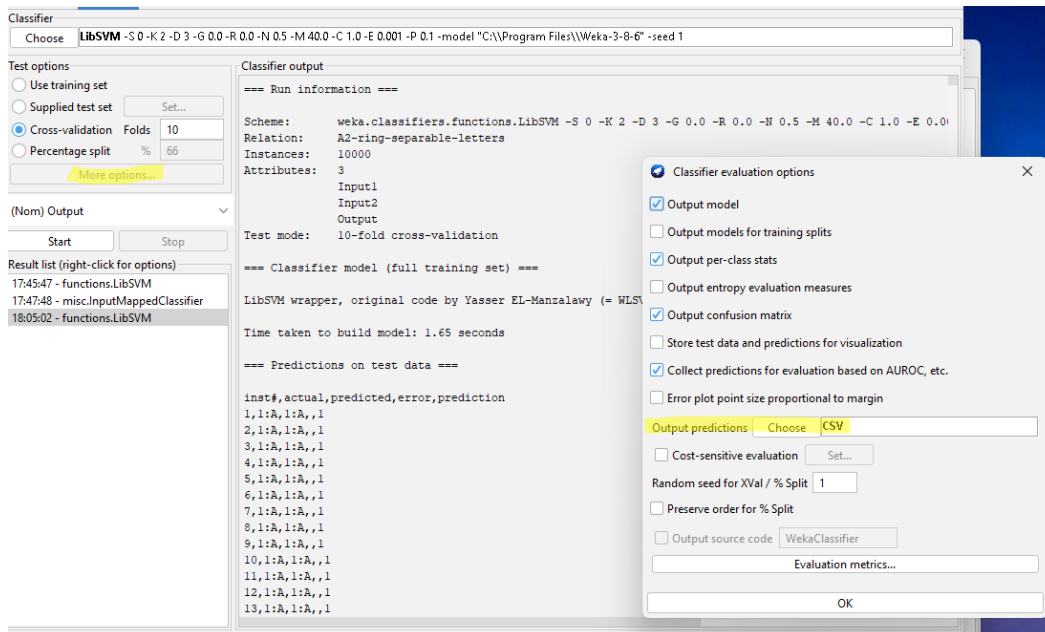


Figure 7: SVM prediction details (separable)

Weka also offers the functionality to display predicted values and visually represent errors. This can be achieved by simply right-clicking on the relevant item in the Result List panel.

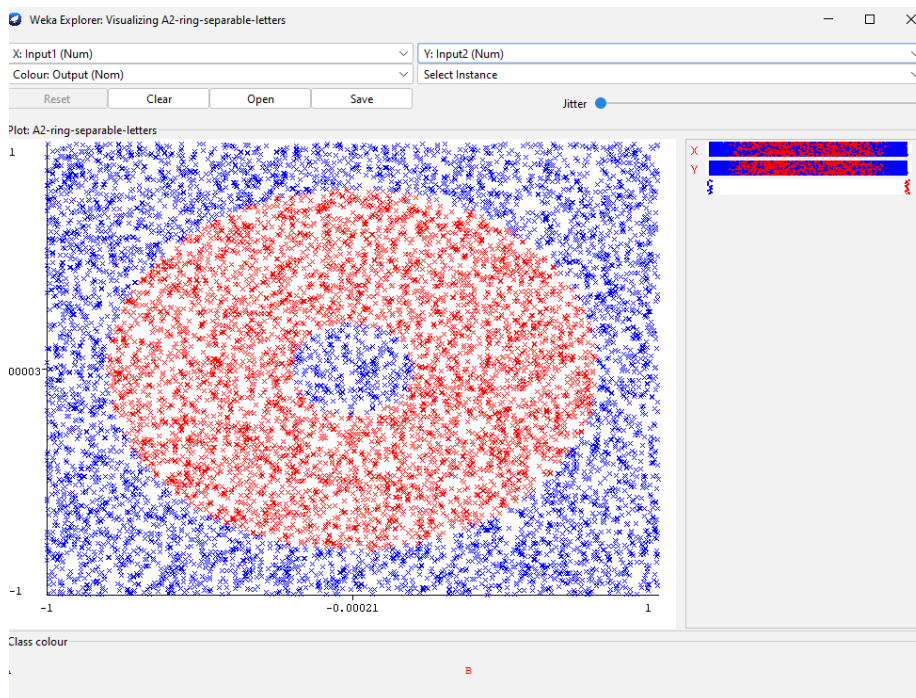


Figure 8: SVM Weka plot result (separable)

Now we will go for "A2-ring-merged-letters.scv"

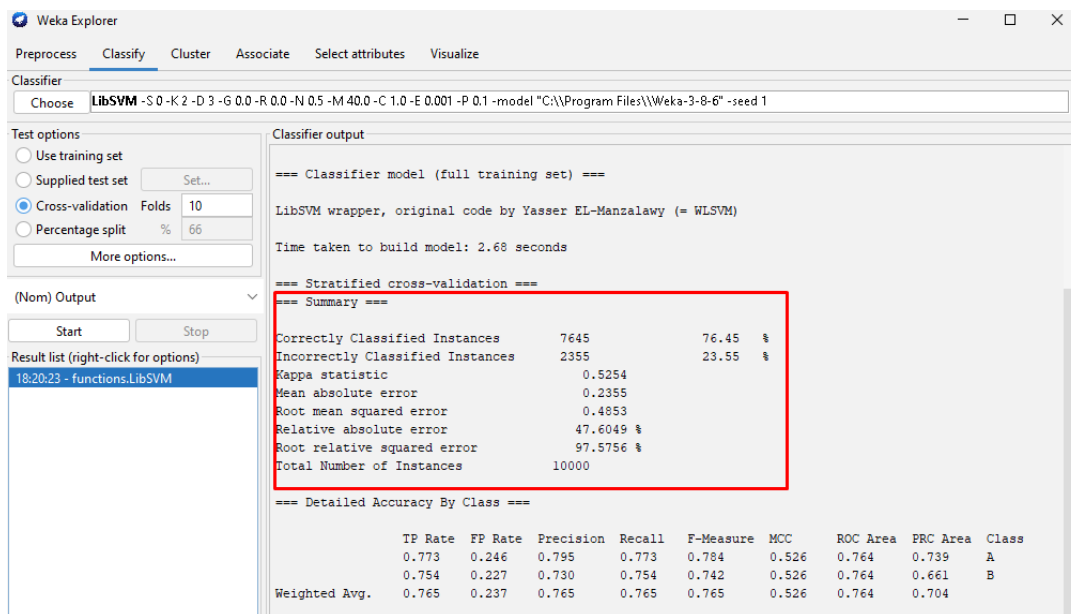


Figure 9: SVM only using training set (merged)

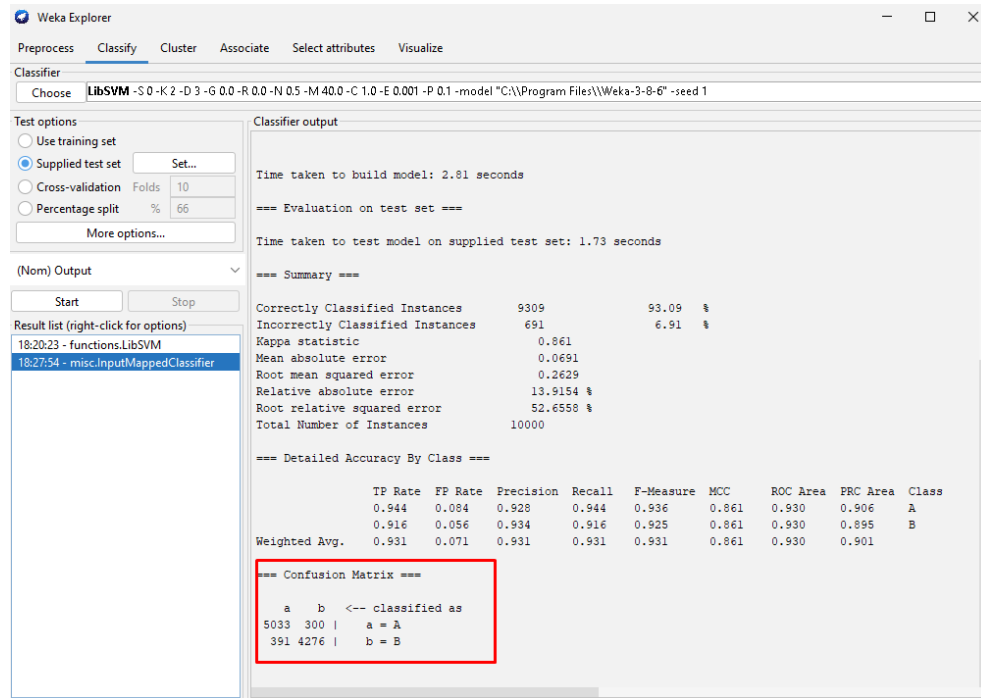


Figure 10: SVM train result using test set (merged)

Now from confusion matrix in Weka let us calculate classification error according to the formula:

$n_{01} = 300$ # False positives

$n_{10} = 391$ # False negatives

$n_{00} = 5033$ # True negatives

$n_{11} = 4276$ # True positives

Classification error formula:

$$E = 100 * (n_{01} + n_{10}) / (n_{00} + n_{01} + n_{10} + n_{11})$$

$$E = 100 (300 + 391 / 5033 + 300 + 391 + 4276)$$

$$E = 100 (691 / 10000)$$

$$E = 100 * 0.0691 = \mathbf{6.91\%}$$

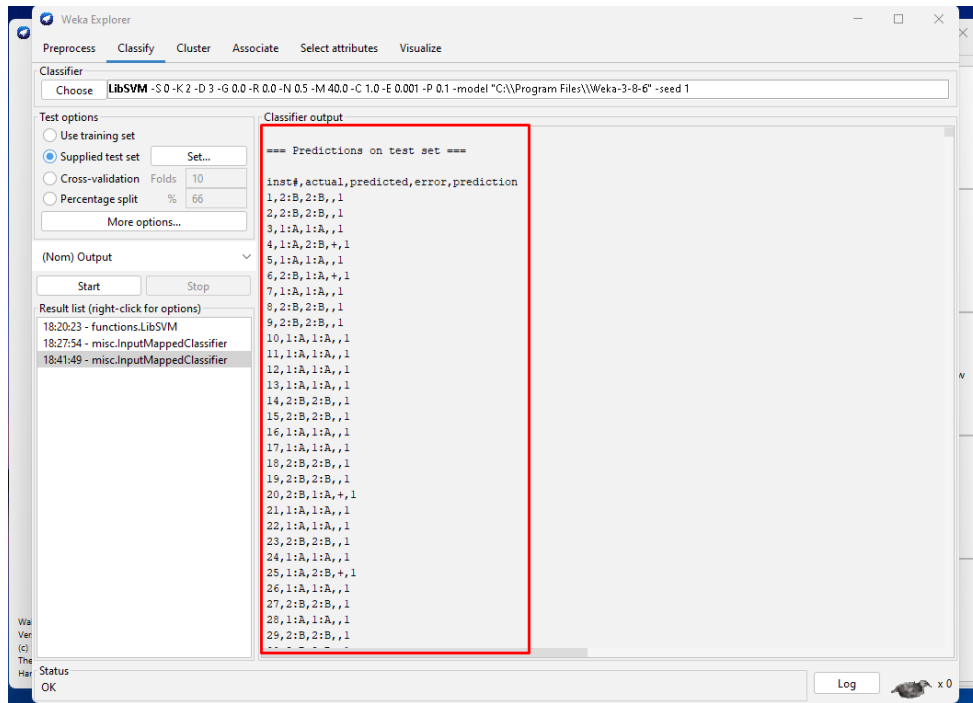


Figure 11: SVM prediction details (merged)

Now let's visualize the result.

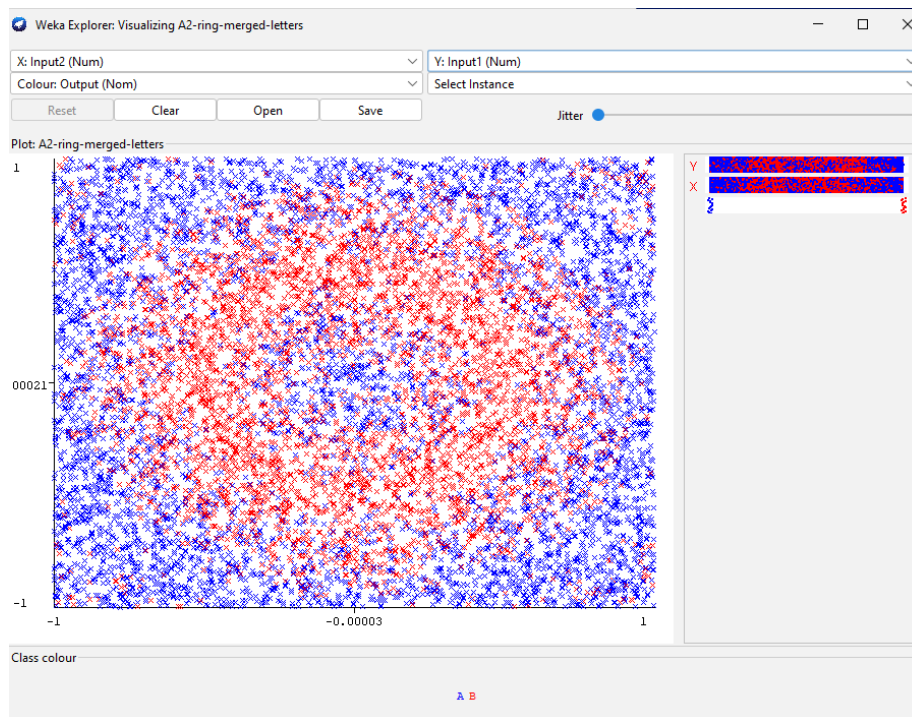


Figure 12: Weka plot result (merged)

Back-Propagation (BP): Back-Propagation (BP) is a key algorithm in neural network training, essential for both neuronal and evolutionary computation. In neuronal computation, it's used to train artificial neural networks by propagating errors back through the network, allowing for efficient adjustment of weights to minimize the loss function. This process involves a forward pass to generate predictions, error calculation, a backward pass to compute the gradient of the error, and weight updates typically using gradient descent. In evolutionary computation, BP often complements evolutionary algorithms, providing precise weight adjustments in hybrid approaches. While evolutionary methods excel in exploring large solution spaces and avoiding local minima, BP offers fine-tuning within local areas for enhanced performance. This integration of BP in evolutionary computation illustrates the synergy between global optimization methods and efficient local search capabilities, critical for solving complex problems and developing adaptive systems.

Back-Propagation (BP), using free software: We have implemented the Back-Propagation (BP) algorithm using TensorFlow in Python, a free and powerful machine learning library. The process began with preparing the dataset: we loaded the training and testing data from CSV files. To ensure the neural network model processes the data effectively, we normalized the input features using the StandardScaler, a common approach for data standardization.

We then set up a neural network model using TensorFlow's Keras API, designing it with an input layer, a hidden layer, and an output layer. The model was compiled with the Adam optimizer and the sparse categorical cross-entropy loss function, suitable for multi-class classification tasks.

To enhance the reproducibility of the model results, we set a random seed. This practice helps to get consistent outcomes across different runs, which is crucial for evaluating model performance and debugging.

The model was trained on the preprocessed training data and then used to make predictions on the test set. We evaluated the model's performance by generating a confusion matrix and calculating the classification error rate. This error rate gives a quantitative measure of how often the model makes incorrect predictions, which is key for understanding and improving the model's accuracy.

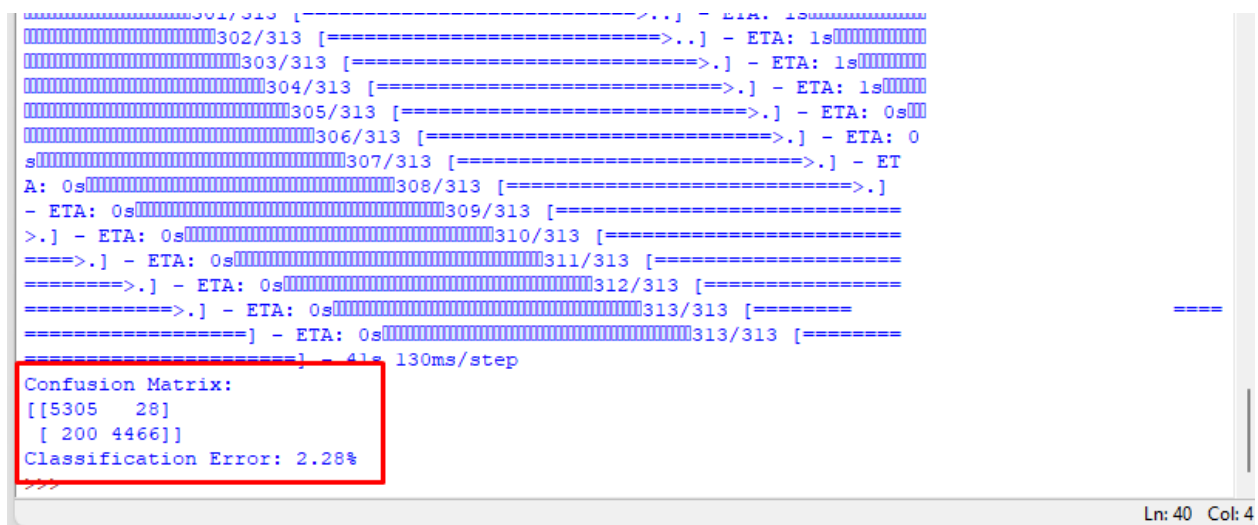


Figure 13: Confusion Matrix and Classification Error (Separable)

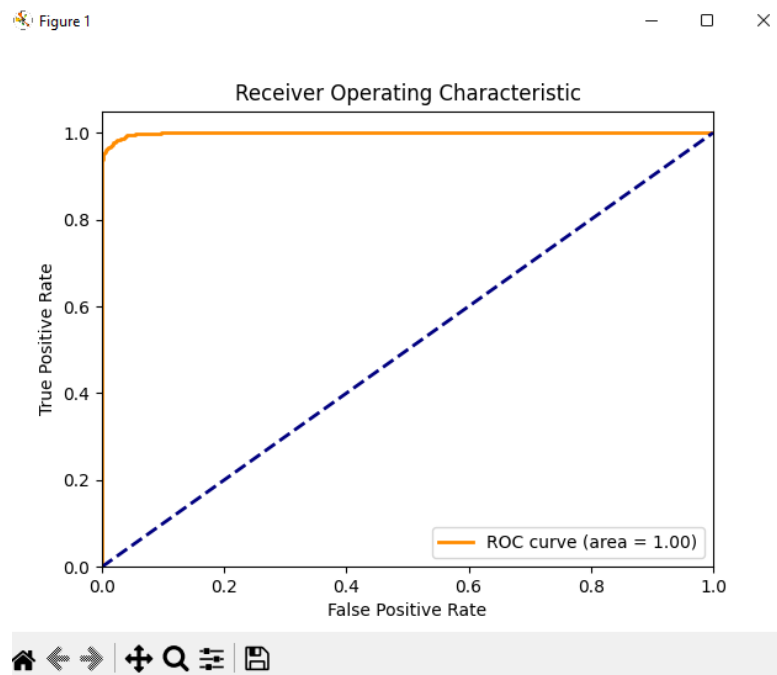


Figure 14: ROC Curve and AUC (Separable)

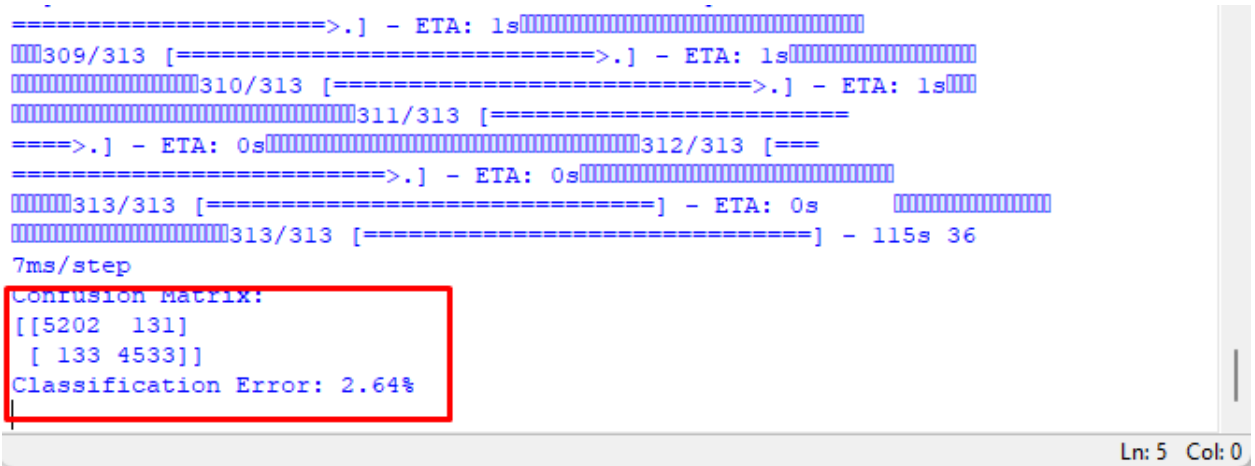


Figure 15: Confusion Matrix and Classification Error (Merged)

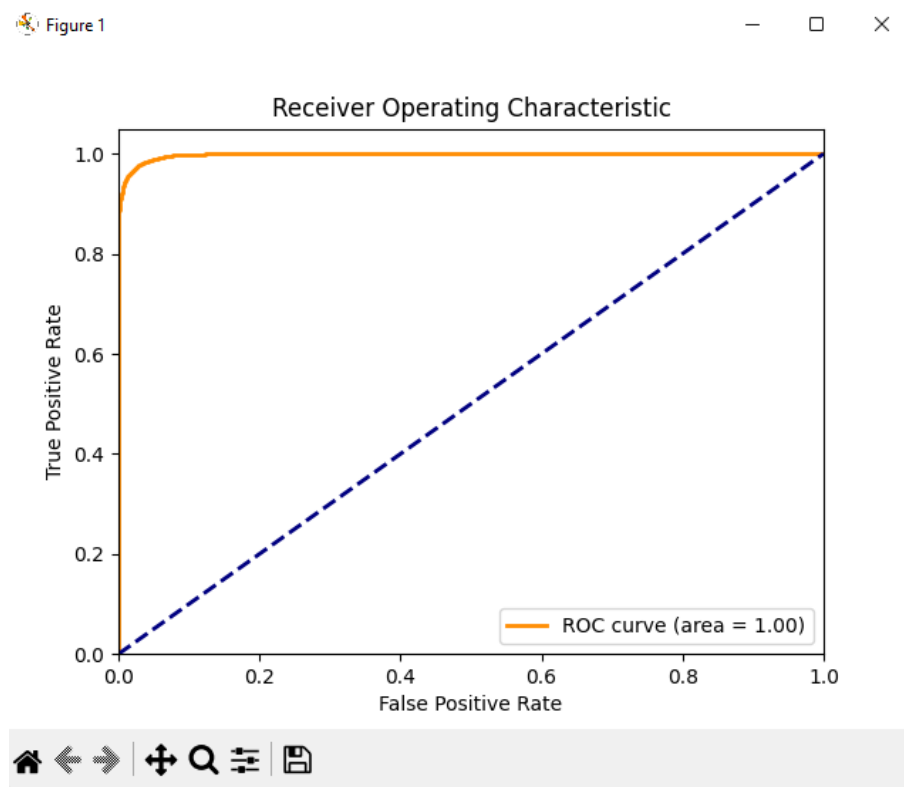


Figure 16: ROC Curve and AUC (Merged)

Multiple Linear Regression (MLR): Multiple Linear Regression (MLR) is a statistical method used to model the relationship between a dependent variable and multiple independent variables. It extends simple linear regression by incorporating more than one predictor. The MLR equation includes coefficients for each independent variable and an intercept, used to predict the outcome variable. MLR is used in various fields for prediction and trend analysis, but it requires certain assumptions like linearity, no multicollinearity, and homoscedasticity to be met for accurate results. The model's fit is evaluated using metrics like R-squared and adjusted R-squared. One of the main challenges in MLR is dealing with multicollinearity among predictors.

Multiple Linear Regression (MLR), using free software: For implementing Support Multiple Linear Regression (MLR), we are going to use a free software called Weka (Waikato Environment for Knowledge Analysis).

First of all, we install the package called leastMedSquared and elasticNet (Figure 17). LibSVM is a separate and widely-used library for Support Vector Machines (SVMs) that provides an efficient and flexible platform for SVM classification and regression.

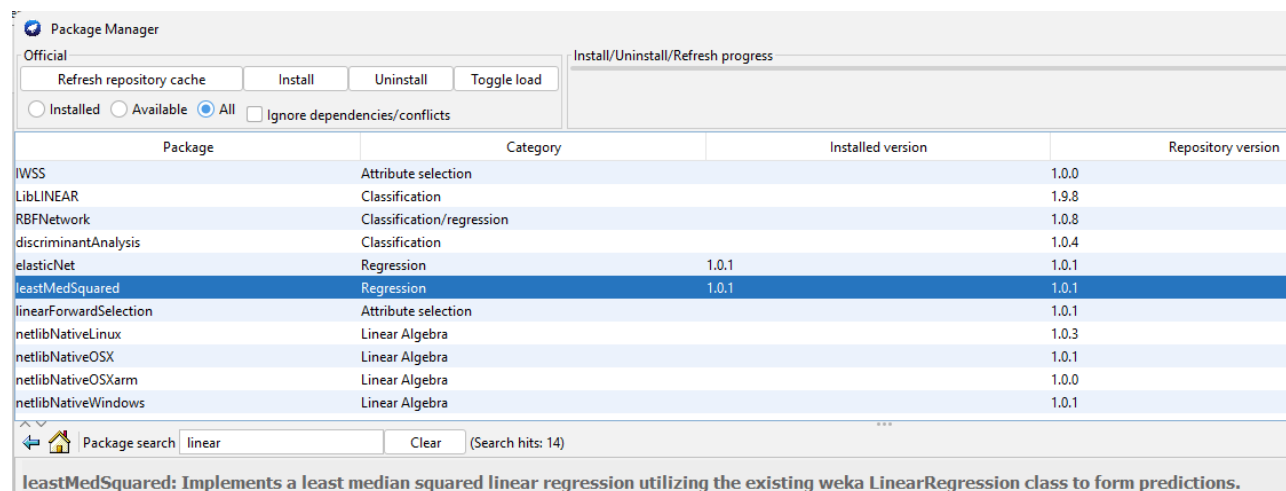


Figure 17: Install leastMedSquared package in Weka

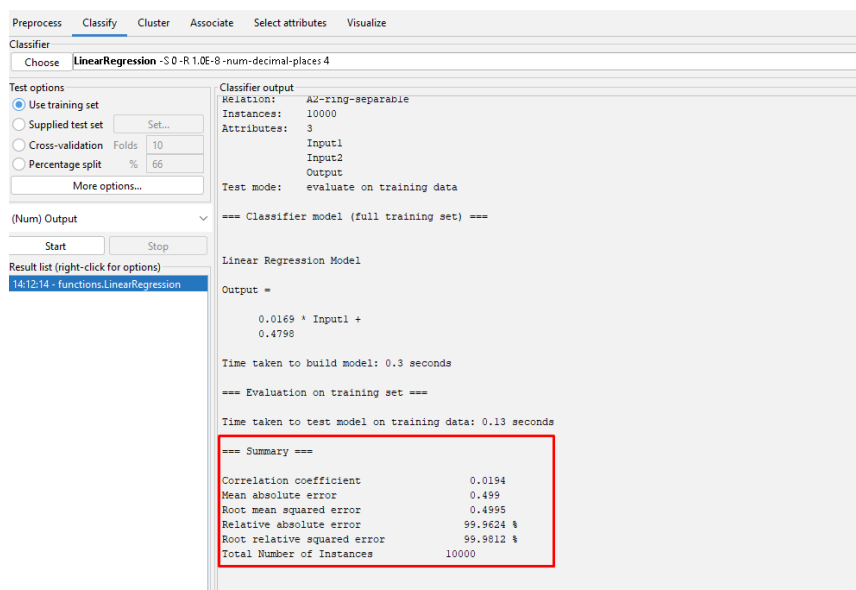


Figure 18: Summary of MLR only using training set (separable)

Next, we conduct another training session similar to the previous one, but this time implement 10-fold cross-validation.

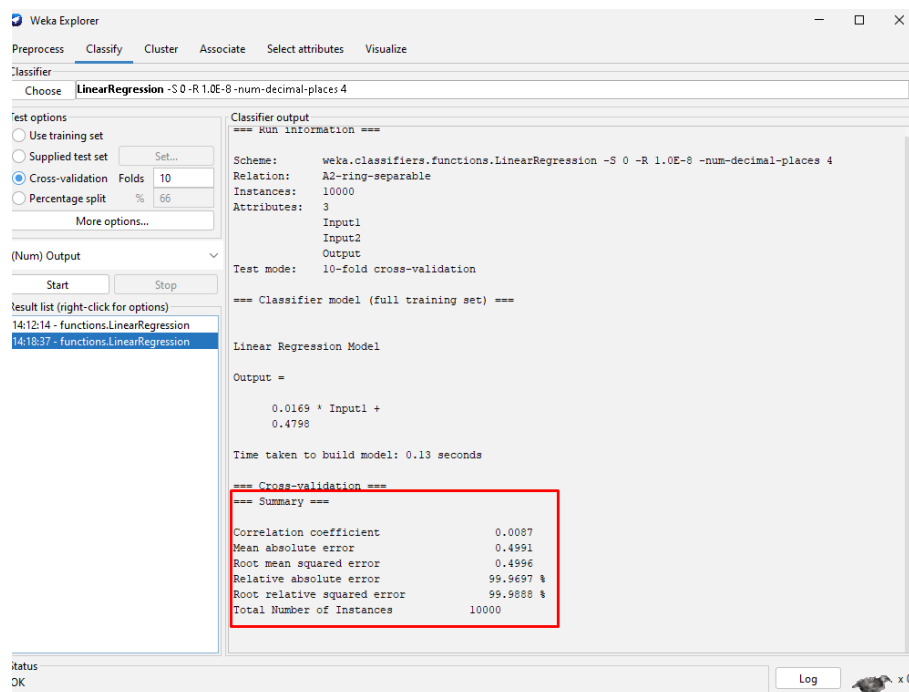


Figure 19: MLR training using 10 folds cross validation (separable)

Subsequently, I initiated a new training session using the test data and proceeded to print out all the predicted values.

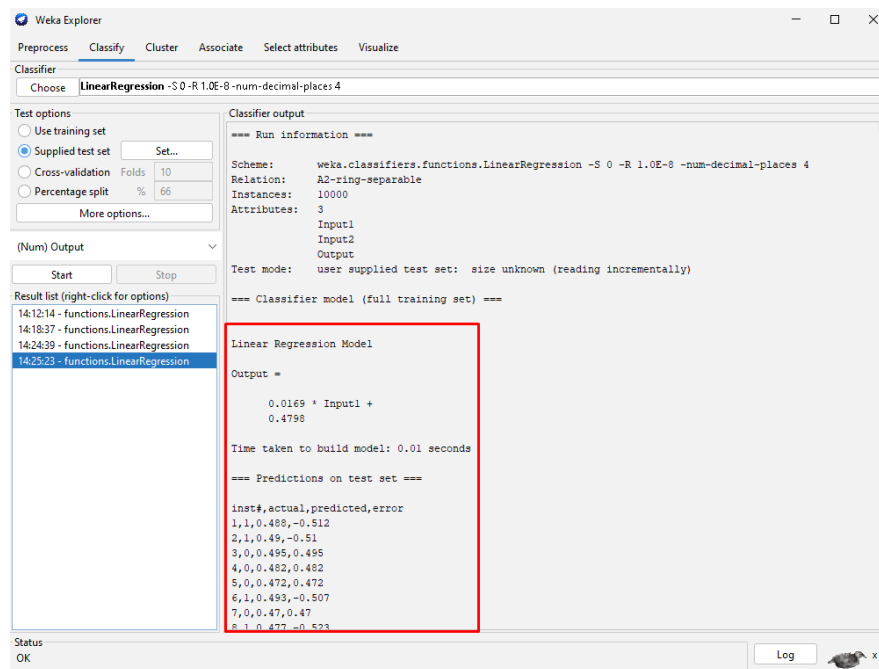


Figure 20: MLR including training data (separable)

I made an Excel sheet to see if the predicted values are above or below the middle value (median), since all the values are less than 0.5. This helps me understand how well the function is approximating.

1	inst	actual	predicted	error		
9987	9986	1	0.468	-0.532		
9988	9987	0	0.47	0.47		
9989	9988	0	0.489	0.489		
9990	9989	0	0.491	0.491		
9991	9990	0	0.47	0.47		
9992	9991	1	0.488	-0.512		
9993	9992	0	0.485	0.485		
9994	9993	1	0.492	-0.508		
9995	9994	1	0.482	-0.518		
9996	9995	0	0.49	0.49		
9997	9996	0	0.496	0.496		
9998	9997	0	0.489	0.489		
9999	9998	0	0.468	0.468		
10000	9999	0	0.495	0.495		
10001	10000	1	0.471	-0.529		
10002		4667				
10003						
10004					sum error	4667
10005					class error	46.67

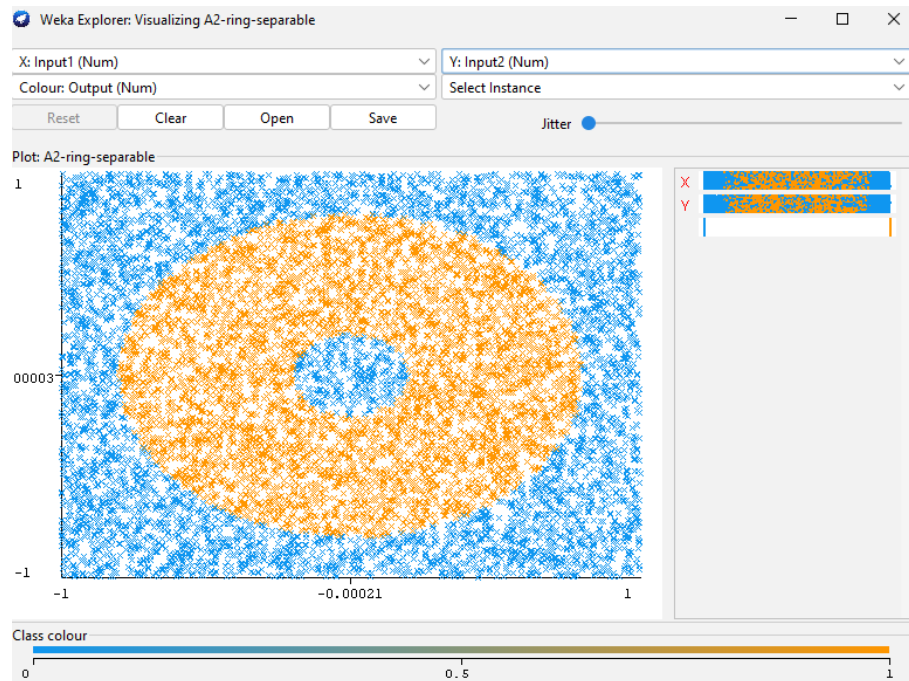


Figure 20: MLR Weka plot result (separable)

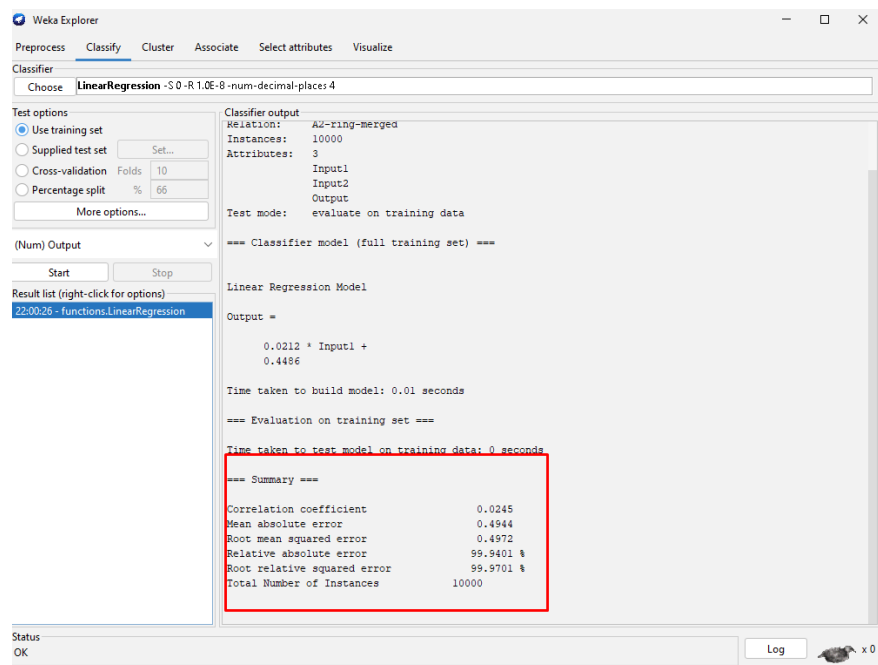


Figure 21: Summary of MLR only using training set (merged)

Next, we conduct another training session similar to the previous one, but this time implement 10-fold cross-validation.

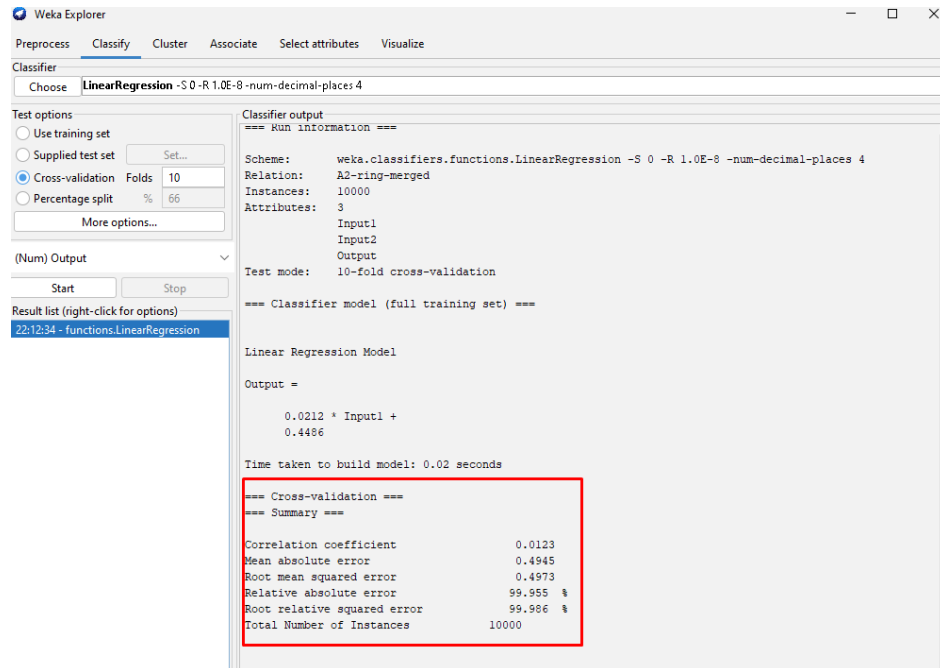


Figure 22: MLR training using 10 folds cross validation (merged)

Now we initiated a new training session using the test data and proceeded to print out all the predicted values.

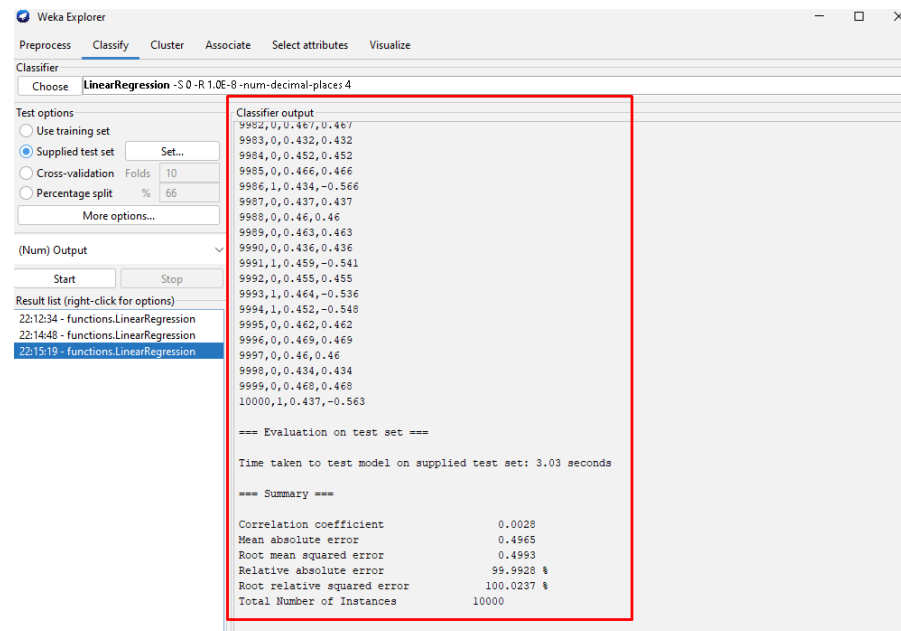


Figure 23: MLR including training data (merged)

1	inst	actual	predicted	error		
9991	9990	0	0.436	0.436		
9992	9991	1	0.459	-0.541		
9993	9992	0	0.455	0.455		
9994	9993	1	0.464	-0.536		
9995	9994	1	0.452	-0.548		
9996	9995	0	0.462	0.462		
9997	9996	0	0.469	0.469		
9998	9997	0	0.46	0.46		
9999	9998	0	0.434	0.434		
10000	9999	0	0.468	0.468		
10001	10000	1	0.437	-0.563		
10002		4667				
10003						
10004					Sum Error	4667
10005					Class Error	46.67

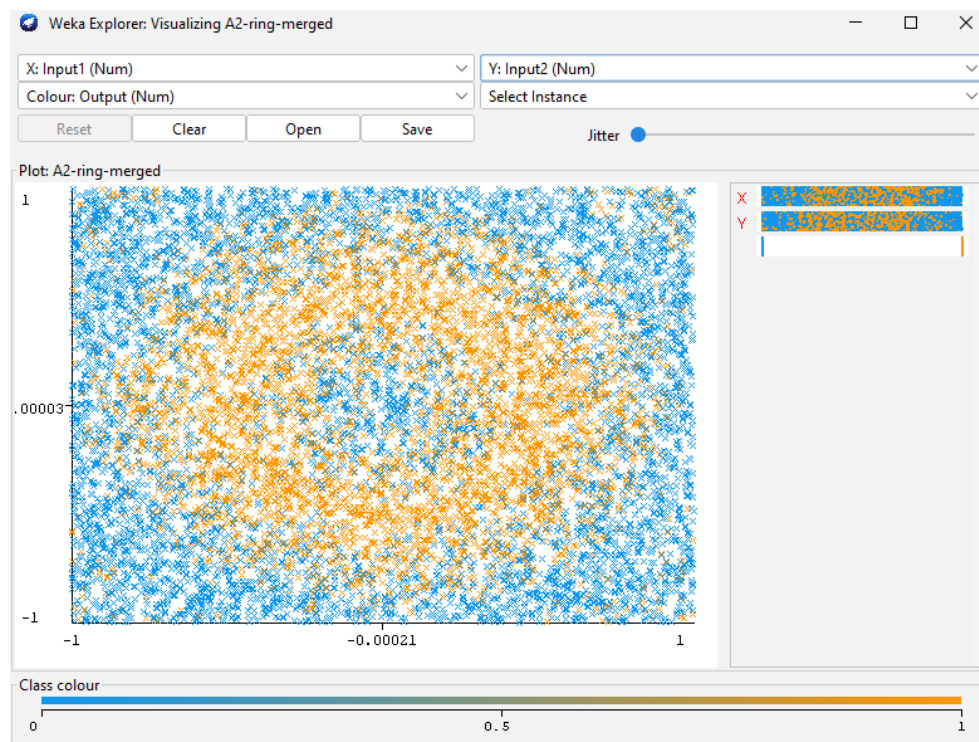


Figure 24: MLR Weka plot result (merged)

Comparison of results: First we create a table with the classification error values obtained using each method:

	SVM	BP	MLR
ring-separable	6.18	2.28	46.67
ring-merged	6.91	2.64	46.67

Here's an interpretation of the results:

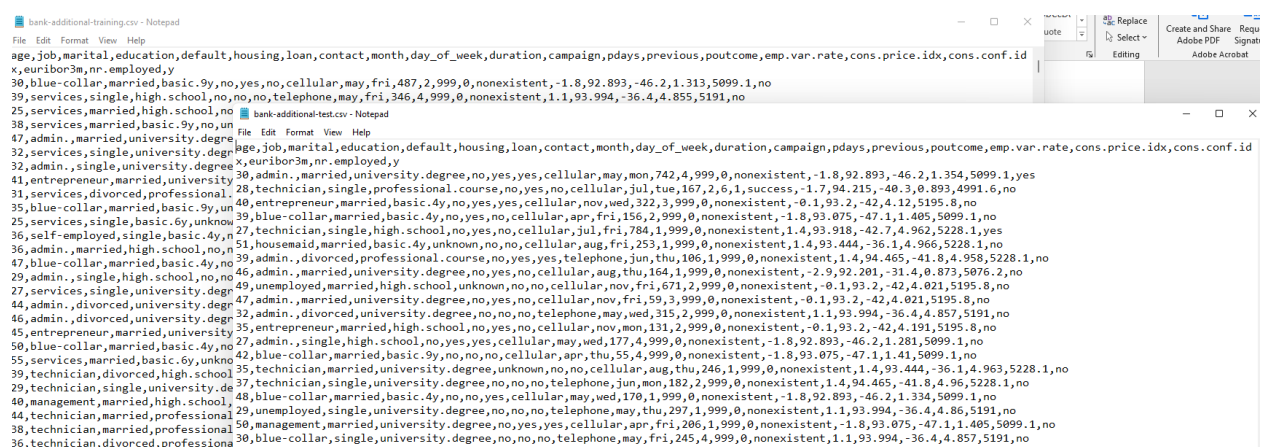
Multiple Linear Regression (MLR): MLR performs poorly in both cases (ring-separable and ring-merged), with a high classification error close to 50%. This suggests that MLR, which is designed to model linear relationships between variables, is not suitable for this problem because the underlying relationship is likely nonlinear.

Backpropagation Neural Network (BP): The BP neural network shows better performance compared to MLR. Although it still has a significant error, it is significantly lower than the MLR error. Neural networks tend to perform well when data is well-separated and can be segmented easily. However, it's noted that the execution time for this method is relatively high, possibly due to the large amount of data.

Support Vector Machines (SVM): SVM outperforms both MLR and BP in terms of classification error. It performs well in both situations (ring-separable and ring-merged). SVM is particularly effective because it can create hyperplanes to separate data points, and in this case, the data can be easily isolated using lines in the plane. Therefore, SVM is considered the winner in this case, with the lowest classification error values and better overall performance.

SVM for Bank Dataset: Before applying machine learning algorithms such as SVM, Back Propagation (BP), or Multiple Linear Regression (MLR), we perform data preprocessing and split the bank dataset into two subsets: a training set and a test set. The dataset is first cleaned, which may involve handling missing values, encoding categorical variables, and addressing any other data quality issues. After cleaning, we shuffle the dataset to ensure that both sets are representative of the overall data distribution.

To create the training and test sets, we use an 80-20 split, where 80% of the data is allocated to the training set, and the remaining 20% is allocated to the test set. This ensures that we have a dedicated portion of the data for model training and another portion for evaluating the model's performance. This split is essential for assessing how well the model generalizes to new, unseen data.



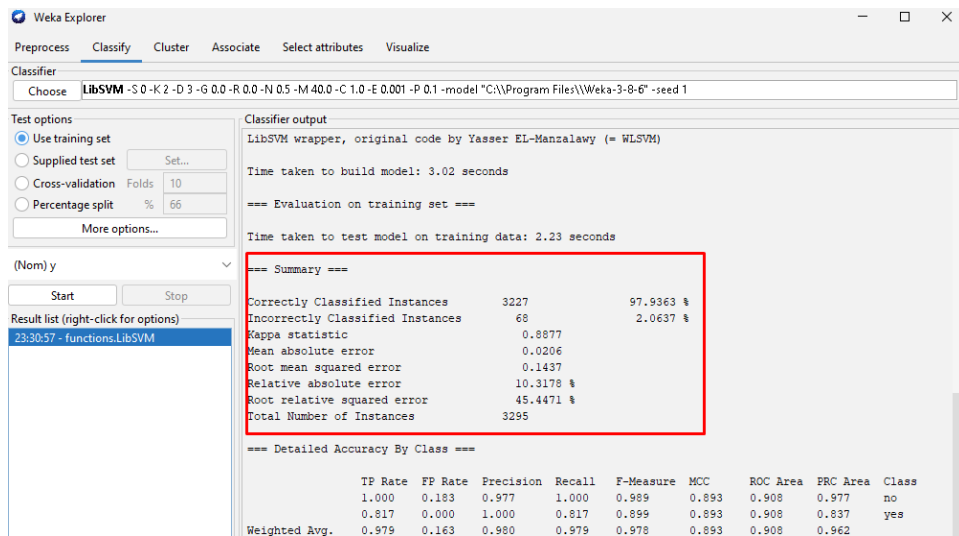


Figure 25: Summary for Training data

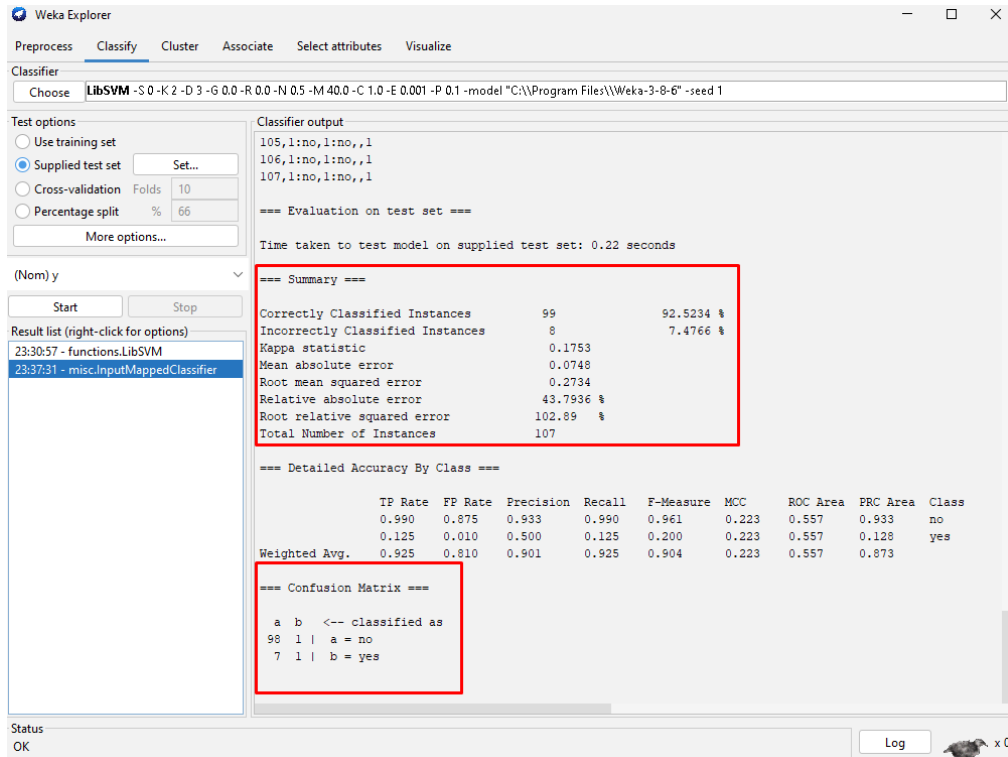


Figure 26: Summary & Confusion Matrix for Training & Test data

From the above confusion matrix, we can calculate the classification error, which is approximately 0.0748 or 7.48%.

BP for Bank Dataset: In this step, we created a supervised machine learning pipeline for a binary classification task using TensorFlow and Keras. We started by loading the training and test datasets from CSV files. Then, we performed data preprocessing, which involved the following steps:

We separated the numeric columns for scaling purposes and used scikit-learn's OneHotEncoder to one-hot encode the categorical columns. Numeric features were standardized to have a mean of 0 and a standard deviation of 1. The target variable was preprocessed to map 'yes' to 1 and 'no' to 0.

Next, we defined a simple feedforward neural network model using Keras. The model consisted of an input layer, a hidden layer with ReLU activation, and an output layer with softmax activation.

We compiled the model with the 'adam' optimizer, 'sparse_categorical_crossentropy' loss function, and accuracy metric. Then, we trained the model on the preprocessed training data for 10 epochs with a batch size of 32.

After training, we used the trained model to make predictions for the test dataset. To assess the model's performance, we computed a confusion matrix and calculated the classification error rate.

```
accuracy: 0.8750 [=====]
[=====] 35/103 [=====>.....] - ETA: 0s -
2.4594 - accuracy: 0.8866[=====]
[=====] 48/103 [=====>.....] - ETA:
s - loss: 2.3939 - accuracy: 0.8750[=====]
[=====] 66/103 [=====>.....]
- ETA: 0s - loss: 2.9099 - accuracy: 0.8617[=====]
[=====] 67/103 [=====>...
.....] - ETA: 0s - loss: 2.8690 - accuracy: 0.8633[=====]
[=====] 83/103 [=====
=====>.....] - ETA: 0s - loss: 2.6044 - accuracy: 0.8697[=====]
[=====] 100/103 [=====
=====>.] - ETA: 0s - loss: 2.4517 - accuracy
[=====] 103/
103 [=====] - 0s 4ms/step - loss: 2.52
8741
1/26 [>.....] - ETA: 4s[=====]
[=====] 26/26 [=====] - ETA: 0s
[=====] 26/26 [=====] - 0s 5
ms/step
Confusion Matrix:
[[744  0]
 [ 80  0]]
Classification Error: 9.71%
>>> |
```

Figure 27: Confusion Matrix and Classification Error with Training & Test data

Here the classification error is approximately 9.71%.

MLR for Bank Dataset: In this part of Multiple Linear Regression (MLR) we used Python with scikit-learn. It starts by setting seed values for random number generators to ensure reproducible results. Next, it loads the training and test datasets from CSV files using pandas. The code separates the dataset into numeric and categorical columns, with numeric features being isolated for scaling. Categorical columns are one-hot encoded to convert them into binary vectors, and this encoding process is applied to both the training and test datasets. The numeric and one-hot encoded categorical features are combined to create feature matrices. The target variable, which contains 'yes' and 'no' values, is preprocessed and mapped to 1 and 0, respectively. Numeric features are standardized to have a mean of 0 and a standard deviation of 1. A Linear Regression model is created and fitted using the training data. Predictions are made on the target variable, and the predicted values are converted to binary based on a threshold. Finally, the code evaluates the model's performance using a confusion matrix and calculates the classification error rate, providing insights into the MLR model's accuracy in predicting the binary target variable.

```
Confusion Matrix:  
[[722  22]  
 [ 52  28]]  
Classification Error: 8.98%  
>>>
```

Figure 28: Confusion Matrix and Classification Error with Training & Test data

Here the classification error is approximately 8.98%.

Comparison of results: First we create a table with the classification error values obtained using each method:

	SVM	BP	MLR
bank-additional	7.48	9.71	8.98

Among these models, the Support Vector Machine (SVM) achieved the lowest classification error rate of 7.48%, indicating the highest level of accuracy in classifying the dataset. On the other hand, the Back-Propagation Neural Network (BP) had a slightly higher error rate of 9.71%, suggesting slightly lower accuracy. The Multiple Linear Regression (MLR) model performed in between, with an error rate of 8.98%. In this context, a lower error rate indicates better model performance, and the SVM model appears to be the most accurate for this particular dataset.

Support Vector Machines (SVM) for Pima Indians Diabetes Dataset: We have downloaded the Pima Indians Diabetes dataset from - <https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names> . For SVM we have changed class variable from numeric to character A for 1 and B for 0.

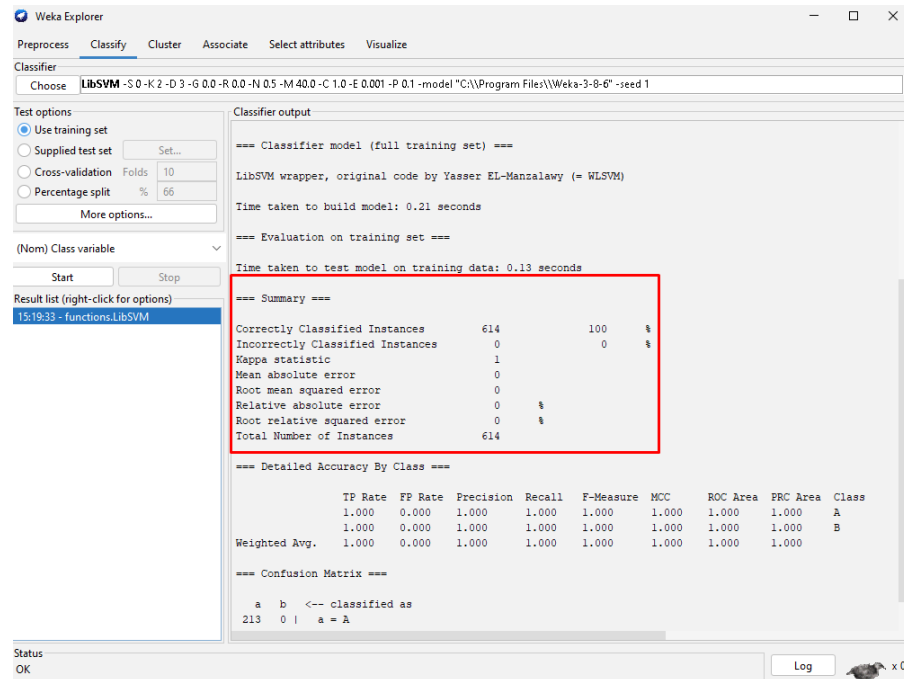


Figure 29: Summary for Training data

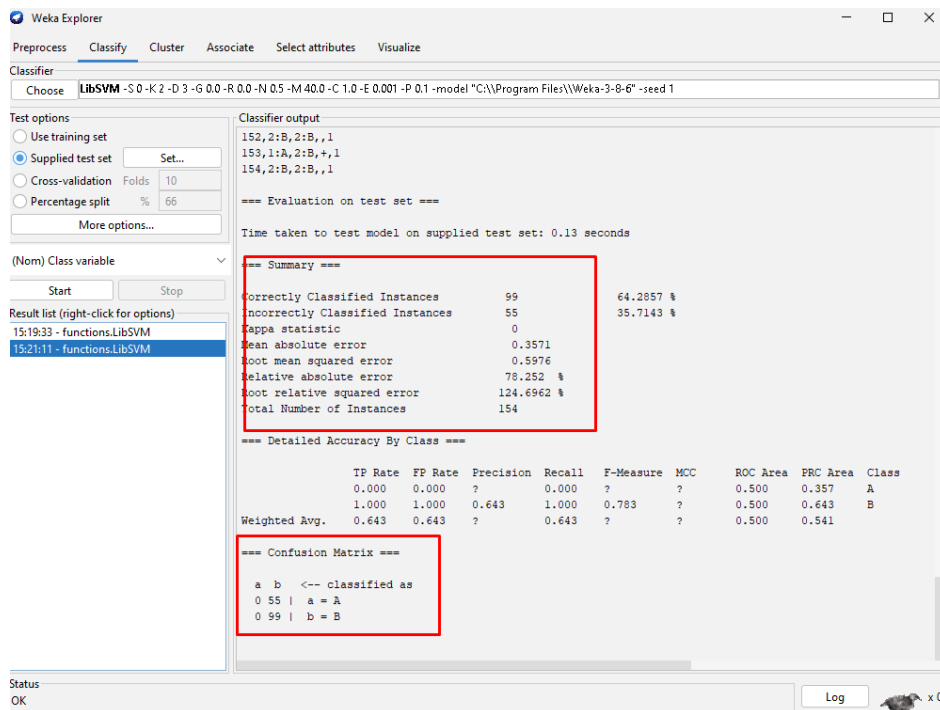


Figure 30: Summary & Confusion Matrix for Training & Test data

From above confusion matrix we get the classification error which is approximately 0.3571, which is equivalent to 35.71%

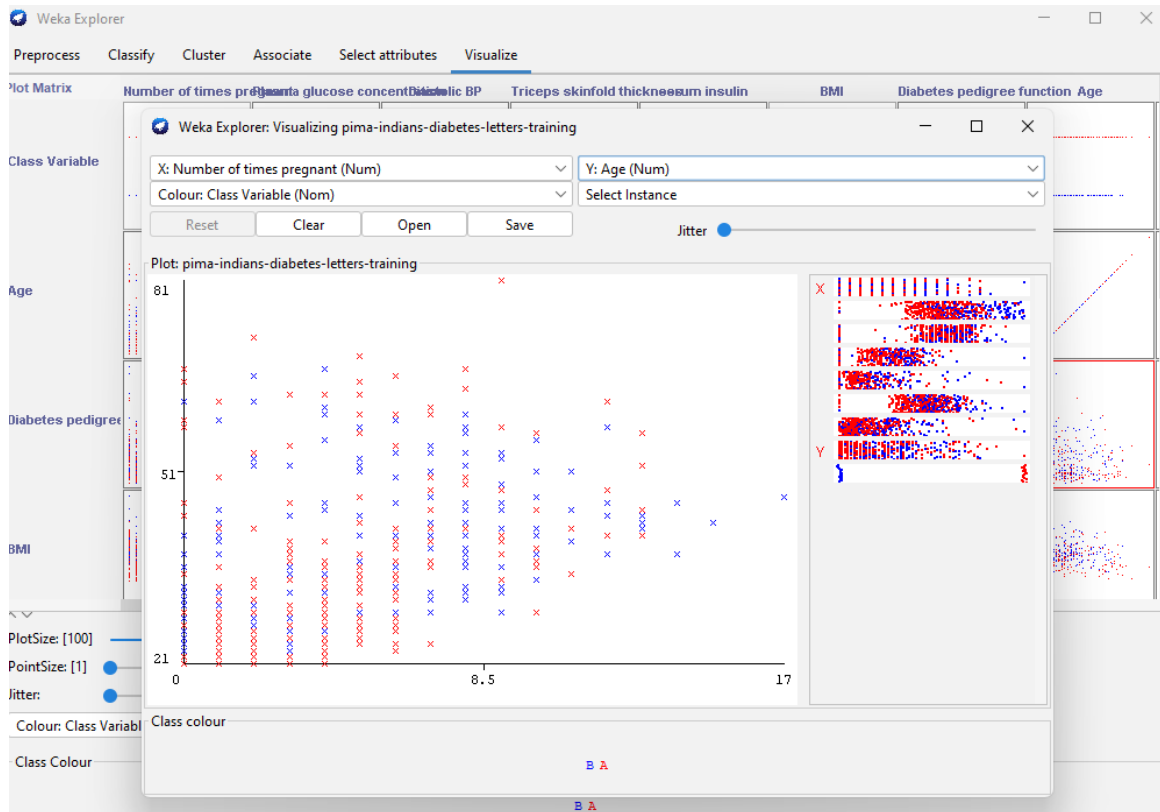


Figure 31: SVM Weka plot result

BP for Pima Indians Diabetes Dataset: In this section we script in python to perform binary classification on two datasets: "pima-indians-diabetes-training.csv" and "pima-indians-diabetes-test.csv." It uses a neural network model built with TensorFlow/Keras. The script loads, preprocesses, and standardizes the datasets, trains the model, predicts outcomes, calculates a confusion matrix, and computes the classification error.


```

Epoch 9/10
 1/20 [>.....] - ETA: 0s - loss: 0.4315 - accurac
500[.....]
16/20 [=====>.....] - ETA: 0s - loss: 0.4150 - accu
: 0.7832[.....]
20/20 [=====] - 0s 4ms/step - loss: 0.4159
accuracy: 0.7899
Epoch 10/10
 1/20 [>.....] - ETA: 0s - loss: 0.2059 - accurac
375[.....]
20/20 [=====] - ETA: 0s - loss: 0.4089 - accu
: 0.8029[.....]
20/20 [=====] - 0s 7ms/step - loss: 0.4089
accuracy: 0.8029
 1/20 [>.....] - ETA: 2s [.....]
20/20 [=====] - 0s 2ms/step
Confusion Matrix:
[[352  49]
 [ 66 147]]
Classification Error: 18.73%
>>> |

```

Figure 32: Summary & Confusion Matrix for Training & Test data

Here the classification error is approximately 18.73%.

MLR for Pima Indians Diabetes Dataset:

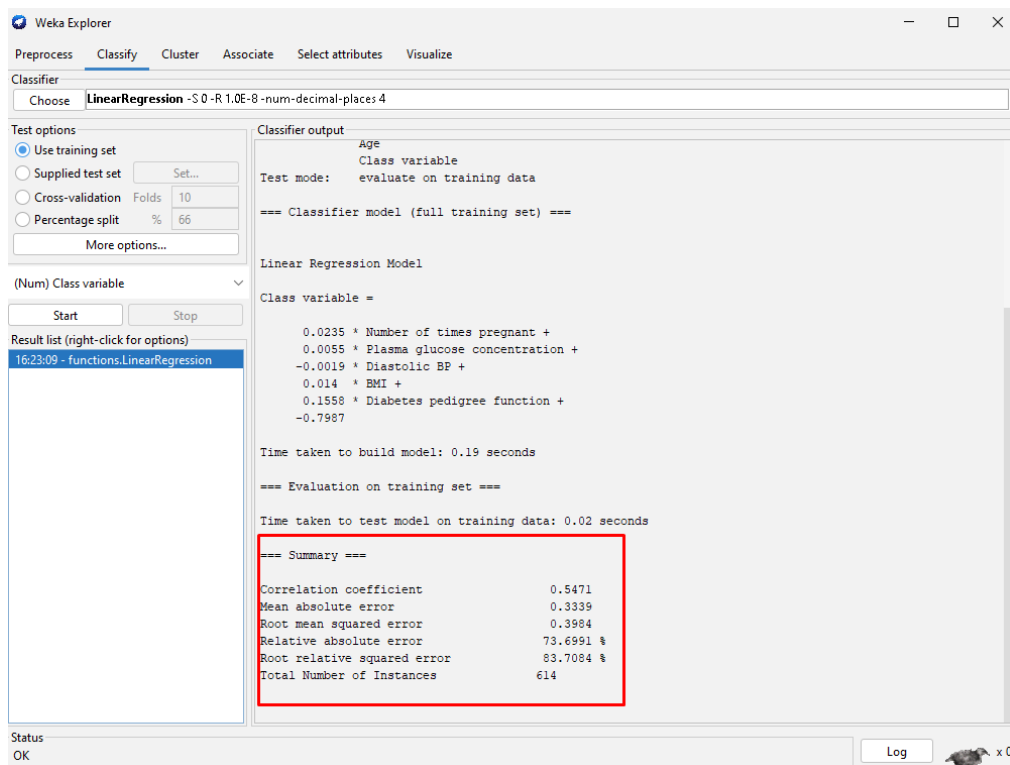


Figure 33: Summary of MLR only using training set

Next, we conduct another training session similar to the previous one, but this time implement 10-fold cross-validation.

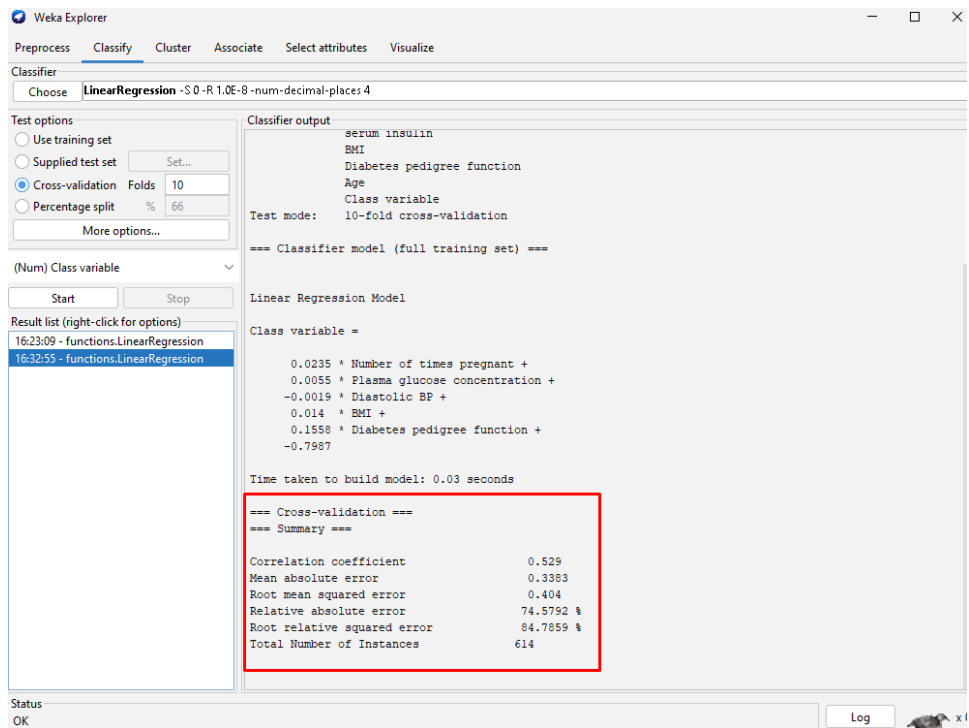


Figure 34: MLR training using 10 folds cross validation

Now we initiated a new training session using the test data and proceeded to print out all the predicted values.

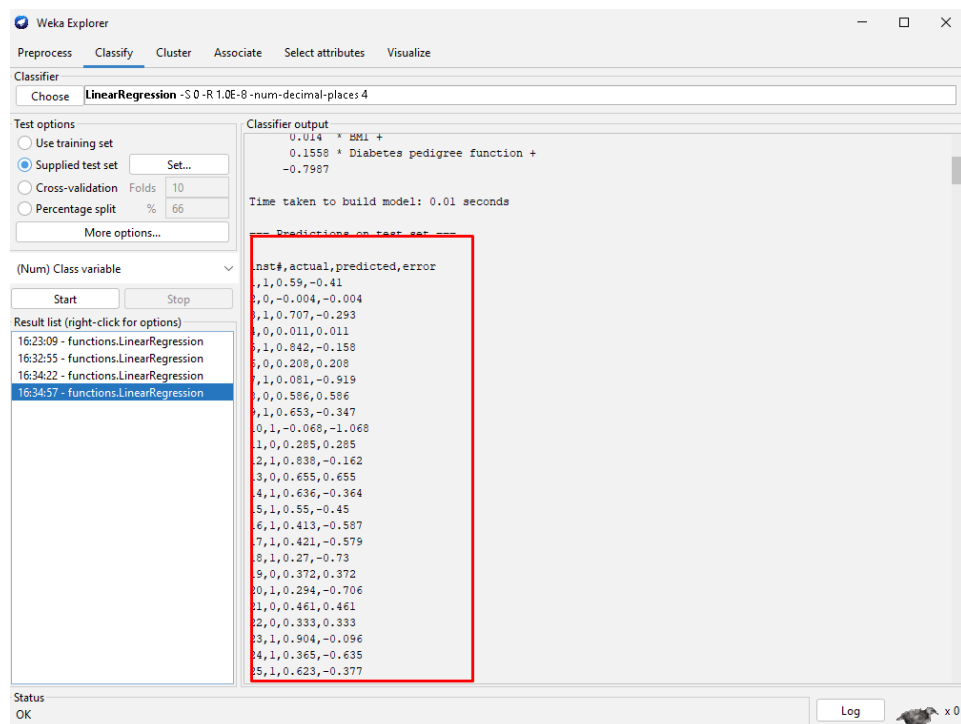


Figure 35: MLR including training data

We made an Excel sheet to see if the predicted values are above or below the middle value (median), since all the values are less than 0.5. This helps me understand how well the function is approximating.

	A	B	C	D	E	F	G
598	597	0	0.093	0.093			
599	598	0	0.147	0.147			
600	599	1	0.568	-0.432			
601	600	0	0.141	0.141			
602	601	0	0.096	0.096			
603	602	0	0.233	0.233			
604	603	0	0.174	0.174			
605	604	1	0.646	-0.354			
606	605	1	0.734	-0.266			
607	606	0	0.377	0.377			
608	607	1	0.831	-0.169			
609	608	0	-0.037	-0.037			
610	609	0	0.508	0.508			
611	610	0	0.077	0.077			
612	611	0	0.233	0.233			
613	612	1	0.674	-0.326			
614	613	1	0.783	-0.217			
615	614	0	0.362	0.362			
616		213					
617						sum error	213
618						class error	2.13

Comparison of results: First we create a table with the classification error values obtained using each method:

	SVM	BP	MLR
pima-indians-diabetes	35.71	18.73	2.13

Among the three models, MLR has the lowest classification error rate (2.13%), followed by BP (18.73%), while SVM has the highest classification error rate (35.71%). This suggests that the MLR model performs the best in terms of accuracy on the "pima-indians-diabetes" dataset, followed by BP, and SVM performs the least accurately.